

FIG. 2A

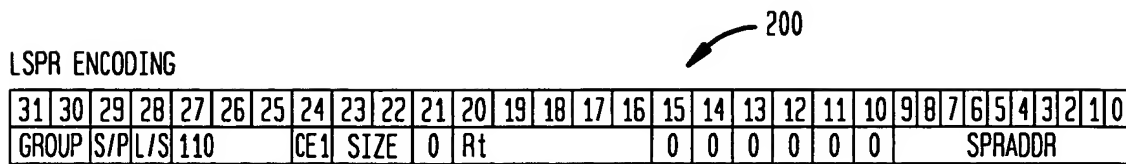


FIG. 2B

Syntax/Operation

Instruction	Operands	Operation
LOAD ADDRESS REGISTER		
LSPR.[SP].W	Rt, SPRADDR	$Rt \leftarrow [SPRADDR]word$
LSPR.[SP].H0	Rt, SPRADDR	$Rt.H0 \leftarrow [SPRADDR]hword$
LSPR.[SP].B0	Rt, SPRADDR	$Rt.B0 \leftarrow [SPRADDR]byte$
T.LSPR.[SP].[WH0B0]	Rt, SPRADDR	Do operation only if T condition is satisfied in F0

FIG. 3A

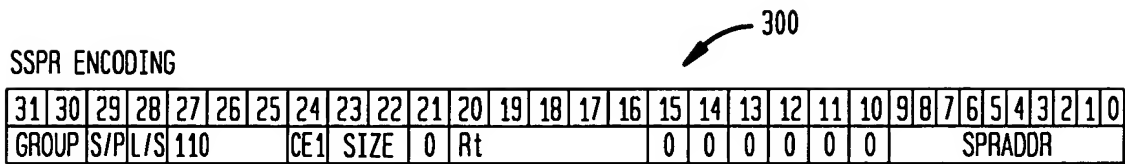
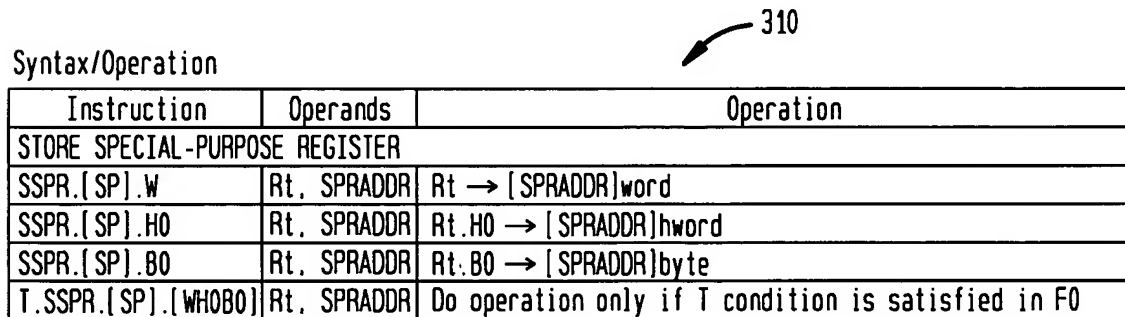


FIG. 3B

Syntax/Operation 

Instruction	Operands	Operation
STORE SPECIAL-PURPOSE REGISTER		
SSPR.[SP].W	Rt, SPRADDR	Rt → [SPRADDR]word
SSPR.[SP].H0	Rt, SPRADDR	Rt.H0 → [SPRADDR]hword
SSPR.[SP].B0	Rt, SPRADDR	Rt.B0 → [SPRADDR]byte
T.SSPR.[SP].[WHOBO]	Rt, SPRADDR	Do operation only if T condition is satisfied in F0

4/28

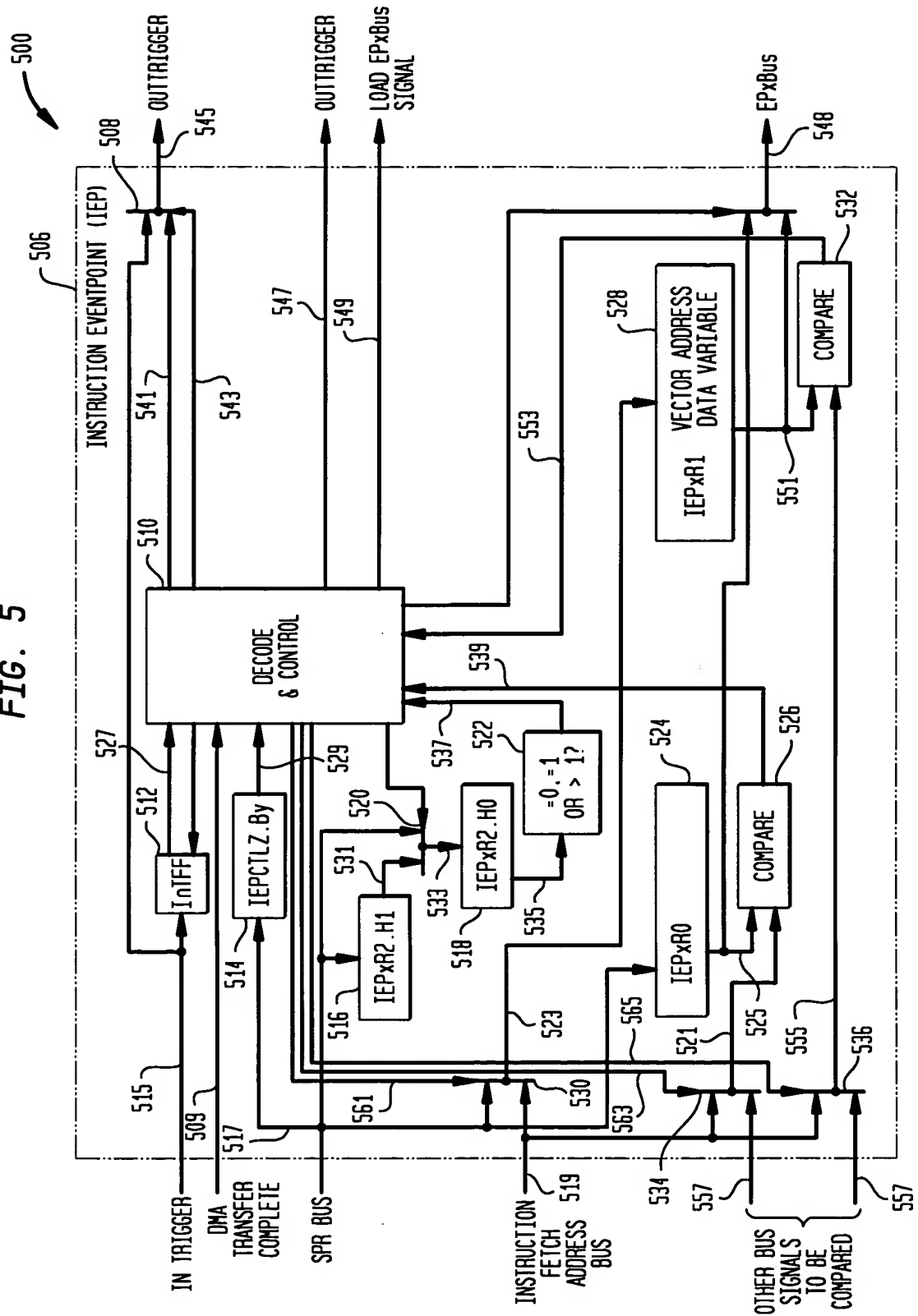
401 403 405 407 SPR REGISTER MAP 409 FIG. 4 400

SYSTEM ADDRESS FOR SP AS A MCB BUS MASTER (THE NON-SP MCB ADDRESS IS THE SAME EXCEPT WITH ADDRESS BIT 22=1)	SP/PE LSPR/ SSPR ADDRESS	SP REGISTER	PE REGISTER	DESCRIPTION
0x0030080	0x0080	DBSTAT		Debug Status register
0x0030084	0x0084	DBIR		Debug Instruction register
0x0030088	0x0088	DBDIN		Debug Data In register
0x003008c	0x008c	DBDOUT		Debug Data Out register
0x0030090	0x0090			
0x0030094	0x0094			
0x0030098	0x0098			
0x003009c	0x009c			
0x00300a0	0x00a0	EPSTAT	EPSTAT	Event Point Status
0x00300a4	0x00a4	DEPCTL0	DEPCTL0	Data Event point Control register 0
0x00300a8	0x00a8	DEPCTL1		Data Event point Control register 1
0x00300ac	0x00ac	DEPOR0	DEPOR0	Data Event point 0 register 0
0x00300b0	0x00b0	DEPOR1	DEPOR1	Data Event point 0 register 1
0x00300b4	0x00b4	DEPOR2	DEPOR2	Data Event point 0 register 2
0x00300b8	0x00b8	DEP1R0		Data Event point 1 register 0
0x00300bc	0x00bc	DEP1R1		Data Event point 1 register 1
0x00300c0	0x00c0	DEP1R2		Data Event point 1 register 2
0x00300c4	0x00c4	DEP2R0		Data Event point 2 register 0
0x00300c8	0x00c8	DEP2R1		Data Event point 2 register 1
0x00300cc	0x00cc	DEP2R2		Data Event point 2 register 2
0x00300d0	0x00d0	IEPCTL0		Instruction Event point Control register 0
0x00300d4	0x00d4	IEPCTL1		Instruction Event point Control register 1
0x00300d8	0x00d8	IEPOR0		Instruction Event point 0 register 0
0x00300dc	0x00dc	IEPOR1		Instruction Event point 0 register 1
0x00300e0	0x00e0	IEPOR2		Instruction Event point 0 register 2
0x00300e4	0x00e4	IEP1R0		Instruction Event point 1 register 0
0x00300e8	0x00e8	IEP1R1		Instruction Event point 1 register 1
0x00300ec	0x00ec	IEP1R2		Instruction Event point 1 register 2
0x00300f0	0x00f0	IEP2R0		Instruction Event point 2 register 0
0x00300f4	0x00f4	IEP2R1		Instruction Event point 2 register 1
0x00300f8	0x00f8	IEP2R2		Instruction Event point 2 register 2
0x00300fc	0x00fc	IEP3R0		Instruction Event point 3 register 0
0x0030100	0x0100	IEP3R1		Instruction Event point 3 register 1
0x0030100	0x0104	IEP3R2		Instruction Event point 3 register 2
0x0030100	0x0108	IEP4R0		Instruction Event point 4 register 0
0x0030100	0x010c	IEP4R1		Instruction Event point 4 register 1
0x0030100	0x0110	IEP4R2		Instruction Event point 4 register 2
0x0030100	0x0114	IEP5R0		Instruction Event point 5 register 0
0x0030100	0x0118	IEP5R1		Instruction Event point 5 register 1
0x0030100	0x011c	IEP5R2		Instruction Event point 5 register 2

410

5/28

FIG. 5



6/28

FIG. 6A

CONTROL VALUE	OPERATION
00000000	Disabled Event point. No action OutTrigger ← InTrigger; 603
<p>604 00T11000</p> <p>This may be used for a loop, with loop skip if count is zero. If T=1 InTrigger can be used to exit or skip the loop.</p>	<p>OutTrigger ← InTrigger; //always pass trigger through 605</p> <p>if(T=1) 606 InTriggerFF ← InTrigger; 607 else 608 InTriggerFF ← 1; 609</p> <p>WHEN(PC=IEPxR0 PC=IEPxR1) 610 { if(PC=IEPxR1)&& ((T=1&&InTriggerFF==1) (IEPxR2.H0==0)) //if PC matches "start" address... 611 //trigger is enabled and active 612 //OR "count" is zero 613) { PC ← IEPxR0; //jump to end of loop 614 IEPxR2.H0 ← IEPxR2.H1; //reload count 615 InTriggerFF ← 0; //clear FF 616 CancelNext(Inst(PC)); //Cancel last next fetched inst 617 //last inst of loop } else if(PC=IEPxR0) //if at "end" address 618 { if(T=1 && InTriggerFF==1) //if trigger enabled and active 619 { PC ← next PC; //fall out of loop 620 IEPxR2.H0 ← IEPxR2.H1; //reload count 621 InTriggerFF ← 0; //clear FF 622 } else if(IEPxR2.H0==0 IEPxR2.H0==1) 623 { PC ← next PC; 624 IEPxR2.H0 ← IEPxR2.H1; 625 } else //else Count is neither 1 nor 0 626 { PC ← IEPxR1; //next PC is [IEPxR1] 627 IEPxR2.H0 ← IEPxR2.H0-1; //decrement "count" 628 } } }</p>

7/28

FIG. 6B

CONTROL VALUE	OPERATION
<p>00T11001</p> <p>This is used for a loop which will not skip to the end if the start address is reached and the count is zero. If T==1 Trigger can be used to exit or skip the loop.</p>	<p>OutTrigger ← InTrigger; //always pass trigger through</p> <p>if(T==1) InTriggerFF ← InTrigger; else InTriggerFF ← 1;</p> <p>WHEN((PC==IEPxR0 PC==IEPxR1)) { if((PC==IEPxR1)&& (T==1&&InTriggerFF==1) //if PC matches "start" address... //trigger is enabled and active) { PC ← IEPxR0; //jump to end of loop IEPxR2.H0 ← IEPxR2.H1; //reload count InTriggerFF ← 0; //clear FF CancelNextInst(PC); //Cancel last next fetched inst //last inst of loop } else if(PC==IEPxR0) //if at "end" address { if(T==1 && InTriggerFF==1) //if trigger enabled and active { PC ← next PC; //fall out of loop IEPxR2.H0 ← IEPxR2.H1; //reload count InTriggerFF ← 0; //clear FF } else if(IEPxR2.H0==0 IEPxR2.H0==1) { PC ← next PC; IEPxR2.H0 ← IEPxR2.H1; } else //else Count is neither 1 nor 0 { PC ← IEPxR1; //next PC is [IEPxR1] IEPxR2.H0 ← IEPxR2.H0-1; //decrement "count" } } }</p>

8/28

FIG. 6C

CONTROL VALUE	OPERATION
<p>00T11010</p> <p>This may be used for a loop with an exit based on a TRUE F0 condition or count == 0, or pre-trigger (if enabled). If enabled for pre-trigger, and trigger FF is set, and 'start' address is matched, then the loop is skipped.</p>	<p>OutTrigger ← InTrigger; //always pass trigger through</p> <p>if(T==1)</p> <p> InTriggerFF ← InTrigger;</p> <p>else</p> <p> InTriggerFF ← 1;</p> <p>WHEN((PC==IEPxR0 PC==IEPxR1))</p> <p>{</p> <p> if((PC==IEPxR1)&& (T==1&&InTriggerFF==1)) //if PC matches "start" address... //trigger is enabled and active</p> <p> {</p> <p> PC ← IEPxR0; //jump to end of loop</p> <p> IEPxR2.H0 ← IEPxR2.H1; //reload count</p> <p> InTriggerFF ← 0; //clear FF</p> <p> CancelNext(Inst(PC)); //Cancel last next fetched inst //last inst of loop</p> <p> }</p> <p> else if(PC==IEPxR0) //if at "end" address</p> <p> {</p> <p> if(T==1 && InTriggerFF==1) //if trigger enabled and active</p> <p> {</p> <p> PC ← next PC; //fall out of loop</p> <p> IEPxR2.H0 ← IEPxR2.H1; //reload count</p> <p> InTriggerFF ← 0; //clear FF</p> <p> }</p> <p> }</p> <p> else if(IEPxR2.H0==0 IEPxR2.H0==1 F0==1)</p> <p> {</p> <p> PC ← next PC;</p> <p> if(IEPxR2.H0==0 IEPxR2.H0==1)</p> <p> {</p> <p> IEPxR2.H0 ← IEPxR2.H1;</p> <p> }</p> <p> }</p> <p> else //else Count is neither 1 nor 0</p> <p> {</p> <p> PC ← IEPxR1; //next PC is [IEPxR1]</p> <p> IEPxR2.H0 ← IEPxR2.H0-1; //decrement "count"</p> <p> }</p> <p>}</p>

9/28

FIG. 6D

CONTROL VALUE	OPERATION
<p>00T11011</p> <p>This may be used for a loop with an exit based on a FALSE F0 condition or count == 0, or pre-trigger (if enabled). If enabled for pre-trigger, and trigger FF is set, and 'start' address is matched, then the loop is skipped.</p>	<p>OutTrigger ← InTrigger; //always pass trigger through</p> <p>if (T==1) InTriggerFF ← InTrigger; else InTriggerFF ← 1;</p> <p>WHEN (PC==IEPxR0 PC==IEPxR1)) { if (PC==IEPxR1) //if PC matches "start" address... (T==1&&InTriggerFF==1) //trigger is enabled and active } { PC ← IEPxR0; //jump to end of loop IEPxR2.H0 ← IEPxR2.H1; //reload count InTriggerFF ← 0; //clear FF CancelNext(Inst(PC)); //Cancel last next fetched inst // (last inst of loop) } } else if (PC==IEPxR0) //if at "end" address { if (T==1 && InTriggerFF==1) //if trigger enabled and active { PC ← next PC; //fall out of loop IEPxR2.H0 ← IEPxR2.H1; //reload count InTriggerFF ← 0; //clear FF } else if (IEPxR2.H0==0 IEPxR2.H0==1 F0==0) { PC ← next PC; if (IEPxR2.H0==0 IEPxR2.H0==1) { IEPxR2.H0 ← IEPxR2.H1; } } else //else Count is neither 1 nor 0 { PC ← IEPxR1; //next PC is [IEPxR1] IEPxR2.H0 ← IEPxR2.H0-1; //decrement "count" } } }</p>

10/28

FIG. 6E

602 CONTROL VALUE	601 OPERATION	670
<p>SPT00000</p> <p>Used for generating an EP interrupt with optional pre-count, and pre-trigger. Interrupt occurs just after instruction at address IEPxR1.</p> <p>If SPT=000, then this option becomes the "No operation" control code and no updates to IEP registers occur.</p>	<pre> if(P==0) //Default is pass-through OutTrigger ← InTrigger; else //Default is "no trigger" OutTrigger ← 0; if(T==1) InTriggerFF ← InTrigger; else InTriggerFF ← 1; WHEN((PC==IEPxR0) (PC==IEPxR1) && (!T InTriggerFF)) { if(T && InTriggerFF) InTriggerFF ← 0; if(IEPxR2.H0 == 0) //If match with count=0, carry on { //acts as if disabled) PC ← next PC; } else if(IEPxR2.H0 == 1) { if(S==1) EPINT ← 1; //assert EP interrupt if(P==1) OutTrigger ← 1; IEPxR2.H0 ← IEPxR2.H1; } else { PC ← next PC; IEPxR2.H0 ← IEPxR2.H0 - 1; } } </pre>	

11/28

FIG. 6F

CONTROL VALUE	OPERATION
OPT00001 Used for vectoring to a target address after 'count' matches	<pre> if(P==0) OutTrigger ← InTrigger; //Default is pass-through else OutTrigger ← 0; //Default is 'no trigger' if(I==1) InTriggerFF ← InTrigger; else InTriggerFF ← 1; WHEN((PC==IEPxR0) && (!I InTriggerFF)) { if(I && InTriggerFF) //If pre-trigger, then clear FF InTriggerFF ← 0; if(IEPxR2.H0 == 0) //If match with count=0, carry on { //acts as if disabled) PC ← next PC; } else if(IEPxR2.H0 == 1) //if Match with count 1 { PC ← IEPxR1; //Branch to vector address IEPxR2.H0 ← IEPxR2.H1; //Reload 'count' } else //if neither 0 or 1... { PC ← next PC; //next PC IEPxR2.H0 ← IEPxR2.H0 - 1; //decrement 'count' } } </pre>

12/28

FIG. 6G

CONTROL VALUE	OPERATION
<p>602</p> <p>SPT00010</p> <p>This IEP can be used to generate an EP interrupt after 'count' input trigger events have been received. (If T==0, then the branch occurs after 'count' cycles, repeatedly)</p>	<p>601</p> <p>690</p> <pre> if(P==0) OutTrigger ← InTrigger; //Default is pass-through else OutTrigger ← 0; //Default is "no trigger" if(T==1) InTriggerFF ← InTrigger; else InTriggerFF ← 1; WHEN(InTriggerFF) { if(T && InTriggerFF) //If pre-trigger, then clear FF InTriggerFF ← 0; if(IEPxR2.H0 == 0) //If match with count=0, carry on { //acts as if disabled PC ← next PC; } else if(IEPxR2.H0 == 1) //if Match with count 1 { if(S==1) EPINT ← 1; //assert EP interrupt if(P==1) OutTrigger ← 1; IEPxR2.H0 ← IEPxR2.H1; } else //If neither 0 or 1... { PC ← next PC; //next PC IEPxR2.H0 ← IEPxR2.H0 - 1; //decrement 'count' } } </pre>

FIG. 7A

ELOOPx
ENCODING

700

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUP		S/P	CtrlOp			LCF	BPID	0	0	0	0	0	0	0	0	0	0	0	0	0	0	UDISP10									

FIG. 7B

Syntax/Operation

710

Instruction	Operands	Operation
ELOOPx	UDISP10	$IEP_{xR1} \leftarrow PC + 1$ $IEP_{xR0} \leftarrow PC + UDISP10$ $IEP_x \leftarrow 0x18$ if($IEP_{xR2}.H0 > 0$) { while($IEP_{xR2}.H0 > 1$) { Execute instructions until $PC = IEP_{xR0}$ $PC \leftarrow IEP_{xR1}$ $IEP_{xR2}.H0 \leftarrow IEP_{xR2}.H0 - 1$ } Execute instructions until $PC = IEP_{xR0}$ $IEP_{xR2}.H0 \leftarrow IEP_{xR2}.H1$ } else $PC \leftarrow PC + UDISP10 + 1$

FIG. 7C

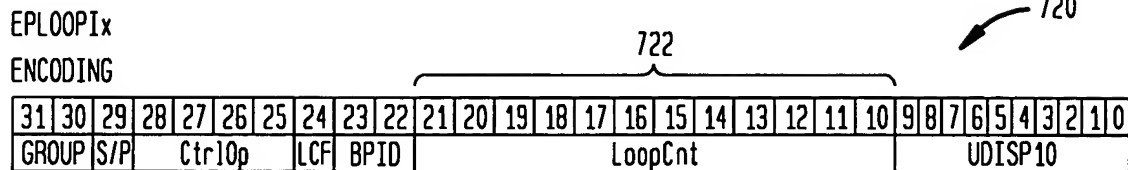


FIG. 7D

Syntax/Operation

Instruction	Operands	Operation
EPL00PIx	LoopCnt, UDISP10,	$IEP_{xR1} \leftarrow PC + 1$ $IEP_{xR0} \leftarrow PC + UDISP10$ $IEP_{xR2.H0} \leftarrow LoopCnt$ $IEP_{xR2.H1} \leftarrow loopCnt$ $IEP_x \leftarrow 0x18$ if ($IEP_{xR2.H0} > 0$) { while ($IEP_{xR2.H0} > 1$) { Execute instructions until $PC = IEP_{xR0}$ $PC \leftarrow IEP_{xR1}$ $IEP_{xR2.H0} \leftarrow IEP_{xR2.H0} - 1$ } Execute instructions until $PC = IEP_{xR0}$ $IEP_{xR2.H0} \leftarrow IEP_{xR2.H1}$ } else $PC \leftarrow PC + UDISP10 + 1$

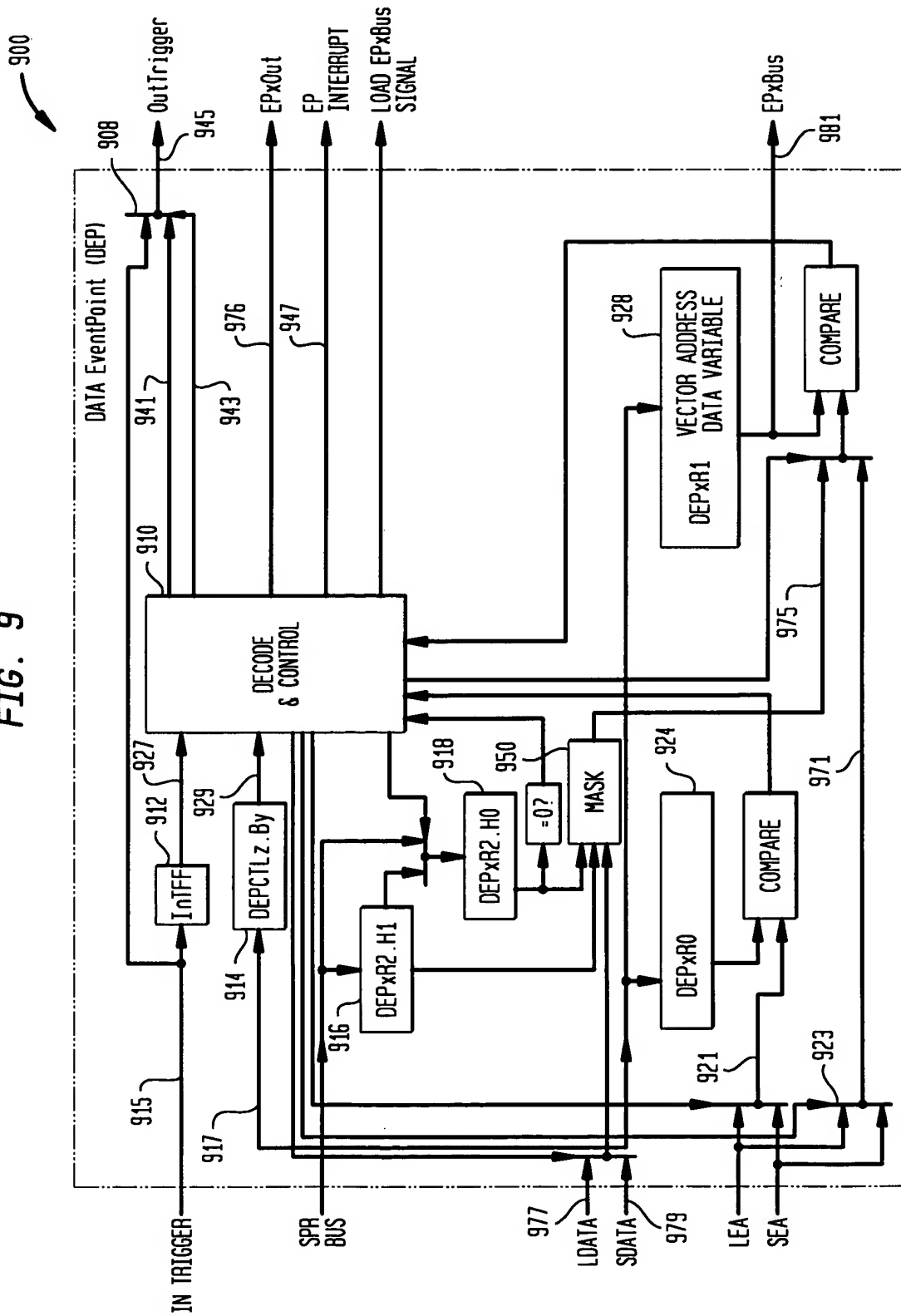
15/28

FIG. 8

800

	802	804	806	808	810
	Cycle	EP Compare	Fetch	Decode	Execute
812	0		ELOOPx	Instruction before ELOOPx	Second instruction before ELOOPx
814	1		First instruction of Loop	ELOOPx 1. Calculate EndA=PC+DISP 2. Hold PC and NOP instruction in fetch	Instruction before ELOOPx
816	2		First instruction of Loop	HW NOP	ELOOPx 1. Send EndA to IEPx on SPR bus to be loaded into IEPxR0 2. Signal IEPx to load the program counter value into IEPxR1 3. Hold PC and NOP instruction in fetch
818	3	First compare of IEPxR0 to PC here.	First instruction of Loop	NOP	HW NOP
820	4		Next instruction	First instruction of Loop	HW NOP
822	5		⋮	Next instruction	First instruction of Loop

FIG. 9



17/28

FIG. 10A

CONTROL VALUE	OPERATION
<p>1012</p> <p>SPT01000</p> <p>LEA match && (LDATA & MASK) match after 1 occurrence then interrupt or OutTrigger</p>	<p>1010</p> <pre> if(P==0) OutTrigger ← InTrigger; //Default is pass-through else OutTrigger ← 0; //Default is "no trigger" if(IT==1) InTriggerFF ← InTrigger; else InTriggerFF ← 1; WHEN((LEA==DEPxR0)&&((LDATA & DEPxR2)==DEPxR1) && InTriggerFF)) { if(IT && InTriggerFF) InTriggerFF ← 0; //if qualified event, clear FF if(P==1) OutTrigger ← 1; //output 1 cycle pulse if(S==1) EPINT ← 1; } </pre>
<p>SPT01001</p> <p>SEA match && (SDATA & MASK) match after 1 occurrence then interrupt or OutTrigger</p>	<p>1015</p> <pre> if(P==0) OutTrigger ← InTrigger; //Default is pass-through else OutTrigger ← 0; //Default is "no trigger" if(IT==1) InTriggerFF ← InTrigger; else InTriggerFF ← 1; WHEN((SEA==DEPxR0)&&((SDATA & DEPxR2)==DEPxR1) && InTriggerFF)) { if(IT && InTriggerFF) InTriggerFF ← 0; //if qualified event, clear FF if(P==1) OutTrigger ← 1; //output 1 cycle pulse if(S==1) EPINT ← 1; } </pre>

FIG. 10B

CONTROL VALUE	OPERATION
<p>1012</p> <p>SPT01010</p> <p>(LEA or SEA) match after count occurrences then interrupt or OutTrigger</p>	<p>1020</p> <pre> if(P==0) OutTrigger ← InTrigger; //Default is pass-through else OutTrigger ← 0; //Default is "no trigger" if(T==1) InTriggerFF ← InTrigger; Else InTriggerFF ← 1; WHEN((LEA==DEPxR0) (SEA == DEPxR1)&&(InTriggerFF)) { if(T && InTriggerFF) //If pre-trigger, then clear FF InTriggerFF ← 0; if(DEPxR2.H0==0) //If match with count=0, carry on { //(acts as if disabled) PC ← next PC; } else if(DEPxR2.H0==1) //if Match with count 1... { if(P==1) OutTrigger ← 1; //output 1 cycle pulse if(S==1) EPINT ← 1; DEPxR2.H0 ← DEPxR2.H1; //Reload 'count' } else //If neither 0 or 1... { PC ← next PC; DEPxR2.H0 ← DEPxR2.H0 - 1; //decrement 'count' } } </pre>

FIG. 10C

CONTROL VALUE	OPERATION
<p>1012</p> <p>SPT01100</p> <p>(LEA THEN SEA) match after count occurrences then interrupt or OutTrigger</p>	<p>1030</p> <pre> if(P==0) OutTrigger ← InTrigger; //Default is pass-through else OutTrigger ← 0; //Default is "no trigger" if(T==1) InTriggerFF ← InTrigger; else InTriggerFF ← 1; WHEN(((LEA==DEPxR0)&&(InTriggerFF))AND THEN(SEA == DEPxR1)) { if(T && InTriggerFF) //If pre-trigger, then clear FF InTriggerFF ← 0; if(DEPxR2.H0==0) //If match with count 0, carry on { //acts as if disabled) PC ← next PC; } else if(DEPxR2.H0==1) //if Match with count 1... { if(P==1) OutTrigger ← 1; //output 1 cycle pulse if(S==1) EPINT ← 1; DEPxR2.H0 ← DEPxR2.H1; //Reload 'count' } else //If neither 0 or 1... { PC ← next PC; DEPxR2.H0 ← DEPxR2.H0 - 1 //decrement 'count' } } </pre>

20/28

FIG. 10D

CONTROL VALUE	OPERATION
<p>1012</p> <p>SPT01101</p> <p>(SEA THEN LEA) match after count occurrences then interrupt or OutTrigger</p>	<p>1040</p> <pre> if(P==0) OutTrigger ← InTrigger; //Default is pass-through else OutTrigger ← 0; //Default is "no trigger" if(T==1) InTriggerFF ← InTrigger; else InTriggerFF ← 1; WHEN(((SEA==DEPxR0)&&(InTriggerFF))AND THEN(LEA == DEPxR1)) { if(T && InTriggerFF) //If pre-trigger, then clear FF InTriggerFF ← 0; if(DEPxR2.H0==0) //If match with count=0, carry on { //acts as if disabled PC ← next PC; } else if(DEPxR2.H0==1) //if Match with count 1... { if(P==1) OutTrigger ← 1; //output 1 cycle pulse if(S==1) EPINT ← 1; DEPxR2.H0 ← DEPxR2.H1; //Reload 'count' } else //If neither 0 or 1... { PC ← next PC; DEPxR2.H0 ← DEPxR2.H0 - 1 //decrement 'count' } } </pre>

FIG. 10E

CONTROL VALUE	OPERATION
<p>1012</p> <p>SPT01110</p> <p>(LEA THEN LEA) match after count occurrences then interrupt or OutTrigger</p>	<p>1050</p> <pre> if(P==0) OutTrigger ← InTrigger; //Default is pass-through else OutTrigger ← 0; //Default is "no trigger" if(T==1) InTriggerFF ← InTrigger; else InTriggerFF ← 1; WHEN(((LEA==DEPxR0)&&(InTriggerFF))AND THEN(LEA == DEPxR1)) { if(T && InTriggerFF) //If pre-trigger, then clear FF InTriggerFF ← 0; if(DEPxR2.H0==0) //If match with count 0, carry on { //acts as if disabled PC ← next PC; } else if(DEPxR2.H0==1) //if Match with count 1... { if(P==1) OutTrigger ← 1; //output 1 cycle pulse if(S==1) EPINT ← 1; DEPxR2.H0 DEPxR2.H1; //Reload 'count' } else //If neither 0 or 1... { PC ← next PC; DEPxR2.H0 ← DEPxR2.H0 - 1 //decrement 'count' } } </pre>

FIG. 10F

CONTROL VALUE	OPERATION
<p>1012</p> <p>SPT01111</p> <p>(SEA THEN SEA) match after count occurrences then interrupt or OutTrigger</p>	<p>1060</p> <pre> if(P==0) OutTrigger ← InTrigger; //Default is pass-through else OutTrigger ← 0; //Default is "no trigger" if(T==1) InTriggerFF ← InTrigger; else InTriggerFF ← 1; WHEN(((SEA==DEPxR0)&&(InTriggerFF))AND THEN(SEA == DEPxR1)) { if(T && InTriggerFF) //If pre-trigger, then clear FF InTriggerFF ← 0; if(DEPxR2.H0==0) //If match with count 0, carry on { //acts as if disabled) PC ← next PC; } else if(DEPxR2.H0==1) //if Match with count 1... { if(P==1) OutTrigger ← 1; //output 1 cycle pulse if(S==1) EPINT ← 1; DEPxR2.H0 ← DEPxR2.H1; //Reload 'count' } else //If neither 0 or 1... { PC ← next PC; DEPxR2.H0 ← DEPxR2.H0 - 1 //decrement 'count' } } </pre>

FIG. 10G

CONTROL VALUE	OPERATION
<p>1012</p> <p>SPT10000</p> <p>LDATA & MASK match after 1 occurrences capture address then interrupt and/or OutTrigger. Once the address is held in DEPXR0, it remains held until the control value changes or a store to DEPXR0 occurs.</p>	<p>1070</p> <pre> if(P==0) OutTrigger ← InTrigger; //Default is pass-through else OutTrigger ← 0; //Default is "no trigger" if(T==1) InTriggerFF ← InTrigger; else InTriggerFF ← 1; DEPXR0 ← LEA; //save address where data detected WHEN((LDATA & DEPXR2) == DEPXR1 && InTriggerFF) { if(T && InTriggerFF) InTriggerFF ← 0; //if qualified event, clear FF Hold(DEPXR0); //Retain state of DEPXR0 (LEA) if(P==1) OutTrigger ← 1; //output 1 cycle pulse if(S==1) EPINT ← 1; } </pre>

FIG. 10H

CONTROL VALUE	OPERATION
<p>1012</p> <p>SPT10001</p> <p>SDATA & MASK match after 1 occurrences capture address then interrupt and/or OutTrigger. Once the address is held in DEPxR0, it remains held until the control value changes or a store to DEPxR0 occurs.</p>	<p>1080</p> <pre> if(P==0) OutTrigger ← InTrigger; //Default is pass-through else OutTrigger ← 0; //Default is "no trigger" if(T==1) InTriggerFF ← InTrigger; else InTriggerFF ← 1; DEPxR0 ← SEA; //save address where data detected WHEN((SDATA & DEPxR2) == DEPxR1 && InTriggerFF) { if(T && InTriggerFF) InTriggerFF ← 0; //if qualified event, clear FF Hold(DEPxR0); //Retain state of DEPxR0 (SEA) if(P==1) OutTrigger ← 1; //output 1 cycle pulse if(S==1) EPINT ← 1; } </pre>

25/28

FIG. 10I

1012

1090

CONTROL VALUE	OPERATION
<p>SPT10010</p> <p>LEA & Mask match.</p> <p>'T' bit used to specify special actions.</p> <p>When T==1, count reg treated as a semaphore which can respond to a DMA write address match. A DMA Write with an address that matches LEA & Mask causes inc of count, while a processor (SP/PE) match event causes dec of count and an EXTOUT pulse.</p> <p>If count==0 when match occurs, no decrement of count occurs and no EXTOUT pulse (basically no action).</p>	<pre> if(P==0) OutTrigger ← InTrigger; //Default is pass-through else OutTrigger ← 0; //Default is "no trigger" to output InTriggerFF ← 1; //Always enable event point for this code //Check for DMA access to increment count if(T==1) if((DMA Write Addr & DEPXR1) == (LEA & DEPXR1)) //compare after DMA mux DEPXR2.H0 = DEPXR2.H0 + 1; //increment count //Check for SP/PE access if((DEPXR0 & DEPXR1) == (LEA & DEPXR1) && InTriggerFF) { if(DEPXR2.H0 == 0) //If match with count=0, carry on { //acts as if disabled //do nothing } else { if(S==1 && P==1) OutTrigger ← 1; //Signal on OutTrigger if(T==0) //if not using DMA signaling... { if(DEPXR2.H0 == 1) // check for count reload { DEPXR2.H0 ← DEPXR2.H1; // reload count if(S==1 && P==0) EPINT ← 1; //if debug int selected, pulse signal } else DEPXR2.H0 = DEPXR2.H0 - 1; //decrement count } else //else Using DMA signaling... { EXTOUT ← 1; Pulse external output 1 cycle DEPXR2.H0 = DEPXR2.H0 - 1; //decrement count } } } </pre>

26/28

FIG. 10J

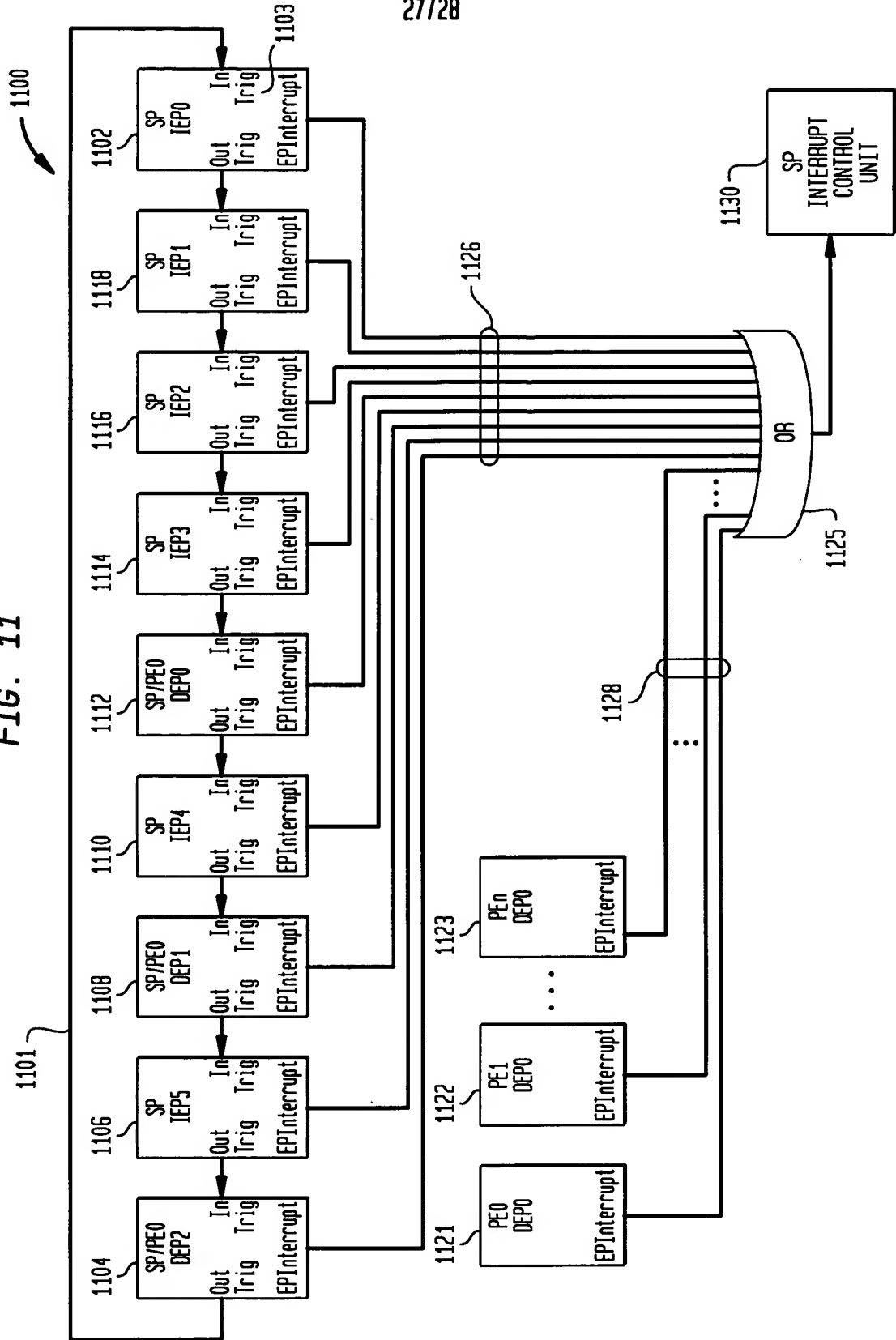
1012

1095

CONTROL VALUE	OPERATION
<p>SPT10011</p> <p>SEA & Mask match.</p> <p>'T' bit used to specify special actions.</p> <p>When T==1, count reg treated as a semaphore which can respond to a DMA write address match. A DMA Read with an address that matches SEA & Mask causes inc of count, while a processor (SP/PE) match event causes dec of count and an EXTOUT pulse. If count==0 when match occurs, no decrement of count occurs and no EXTOUT pulse (basically no action).</p>	<pre> if(P==0) OutTrigger ← InTrigger; //Default is pass-through else OutTrigger ← 0; //Default is "no trigger" to output InTriggerFF ← 1; //Always enable event point for this code //Check for DMA access to increment count if(T==1) if((DMA Read Addr & DEPXR1) == (SEA & DEPXR1) //compare after DMA mux DEPXR2.H0 = DEPXR2.H0 + 1; //increment count //Check for SP/PE access if((DEPXR0 & DEPXR1) == (LEA & DEPXR1) && InTriggerFF) { if(DEPXR2.H0 == 0) //If match with count=0, carry on { //acts as if disabled //do nothing } else { if(S==1 && P==1) OutTrigger ← 1; //Signal on OutTrigger if(T==0) //if not using DMA signaling... { if(DEPXR2.H0 == 1) // check for count reload { DEPXR2.H0 ← DEPXR2.H1; // reload count if(S==1 && P==0) EPINT ← 1; //if debug int selected, pulse signal } else DEPXR2.H0 = DEPXR2.H0 - 1; //decrement count } else //else Using DMA signaling... { EXTOUT ← 1; Pulse external output 1 cycle DEPXR2.H0 = DEPXR2.H0 - 1; //decrement count } } } } </pre>

27/28

FIG. 11



28/28

FIG. 12A

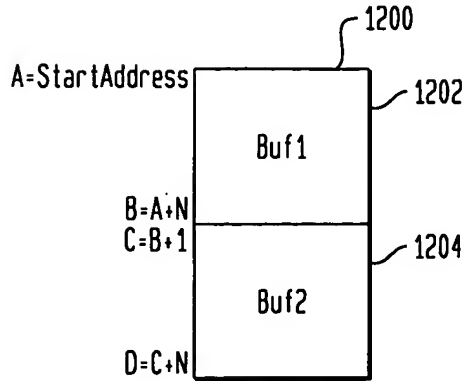


FIG. 12B

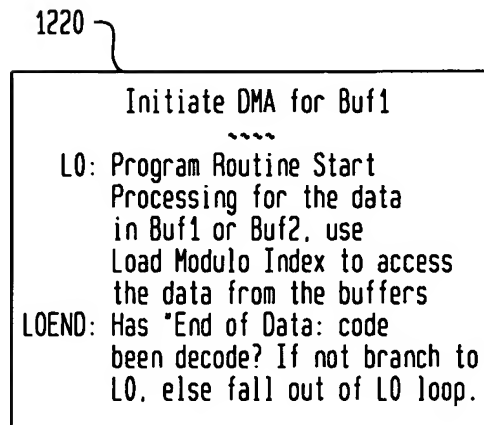


FIG. 12C

